Spring 5-1-2009

Linear system solver scalability for applications of the bidomain cardiac simulations

Scott Matthew Busch
University of Colorado Boulder

Linear System Solver Scalability for Applications of the Bidomain Cardiac Simulations

by

Scott Matthew Busch

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Bachelor of Science

Department of Computer Science

2008

This thesis entitled:
Linear System Solver Scalability for Applications of the Bidomain Cardiac Simulations
written by Scott Matthew Busch
has been approved for the Department of Computer Science

Xiao-Chuan, Cai

Elizabeth Jessup

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Busch, Scott Matthew (B.S. Computer Science)

Linear System Solver Scalability for Applications of the Bidomain Cardiac Simulations

Thesis directed by Processor and Chair Xiao-Chuan, Cai

Contents

Tables

Table

Figures

Figure

Chapter 1

Introduction

The Bidomain problem consists of a set of nonlinear equations with the intent of modeling the electrical properties of the heart tissue. They are designed to model how current and voltage are able to propagate through a sample of cardiac tissue cells. To solve these equations, a nonlinear system solver uses a linear system solver to slowly iterate towards a final solution to the nonlinear equations. A goal of our work is to determine how well the implementation of the Bidomain problem is able to efficiently use additional computational resources. The scalability of the overall Bidomain problem is greatly weighed on the underlying linear solver. Because the majority of the time required to find a solution to the problem is spent in this component, studying the scalability of the linear solver provides insight into how well the Bidomain problem performs with more processors and memory.

The linear system solver that is being investigated for our research is the Scalable Linear System Solver (KSP) that is provided by the Portable, Extensible Toolkit for Scientific Computing (PETSc) [2]. PETSc provides all the necessary tools to solve a system of nonlinear equations and has many algorithm choices available for controlling the system solvers.

For our research, two different problems are being used to assist with measuring the scalability of KSP: the Bidomain problem and a Poisson example. Through the use of these two example problems, KSP is tested using two different linear system solving algorithms and therefore compared. There are many additional parameters of the system solvers that can be specified and have been varied to determine their impacts on the scalability. The Bidomain problem is defined over a two dimensional tissue sample for simulating the heart tissue that evolves over time. The Poisson problem is defined over a similar two dimensional domain. The details of these two problems including their implementations and parameters are discussed in future sections.

To measure scalability, we use two measures: the speedup and the ratio of the time spent performing the communication versus the computational components. Using these two measures, it is possible to evaluate how well the linear system solver is able to find a solution to the desired problems. The work and results of our research have been included and described throughout the sections of this report.

Chapter 2

Background

Today, mathematical models of the physical phenomenon taking place all around us are being used to assist with much of the testing for new theories and technological advances. Among the medical community, models are being developed and tested for reliability of simulating how the many parts of the body function and behave. One such area of interest is the development of a model that is capable of describing the functioning of the heart and the cardiac tissue cells. Many of these mathematical models are very expensive computationally and therefore may require many resources and time in order to complete [4, 6, 7, 9, 14]. Part of developing these mathematical models is invested in understanding the software tools that are available to solve the systems. Knowing how well the tools can operate with more resources will set the direction of focus for the types of problems that can use those tools [12].

2.1 The Bidomain Cardiac Excitation Model

One of the most common and accepted models being studied and in use today is the Bidomain model [4, 8]. The Bidomain model is a system of coupled partial differential equations (PDEs) that are able to mathematically describe the electrical currents and voltage differences of the cardiac tissue [4, 8]. The Bidomain model consists of two nonlinear equations: an elliptic and parabolic PDE [7, 9, 14].

For the Bidomain model, the cardiac tissue is considered to be a combination of an intracellular and extracellular domain [4, 7, 8, 9, 14]. These two domains are separated by the transmembrane [8]. Through the interaction of these three regions, voltage differences are responsible for generating and driving a waveform that propagates through the tissue sample [9]. The two equations of the Bidomain model are a parabolic equation for the transmembrane voltage (V) and an elliptic equation modeling the extracellular voltage ($\mathbf{U}$) [9]. The system can be expressed as

$$\chi c_m \frac{\partial V}{\partial t} + I_{ion} - I_{app} = -\nabla \cdot (D_e \nabla U) \ ,$$

$$\nabla \cdot (D_i \nabla V) = -\nabla \cdot (D \nabla U) \ ,$$

where χ and c_m are the surface to volume ratio and membrane capacitance, respectively [4, 7, 9]. I_{ion} and I_{app} can be either constants or functions that represent the ionic current model being used and any external current applied to the system, respectively [7, 9]. Finally, D, D_i, and D_e are the bulk, intracellular, and extracellular conductivity tensors, respectively, used to describe the direction and intensity of current transfer from one cell to its neighbors, where $D = D_i + D_e$. By using different variations of the ionic current parameter, the complexity of the system can vary greatly and the number of parameters needed can grow quickly [7].

The Bidomain model as it exists today is derived from Hodgkin and Huxley's mathematical model of the axon nerve cell from the 1950's as mentioned in References [7, 9] and references therein. However, as the model attempts to become more realistic and include more aspects, it also becomes a larger system of coupled equations encompassing many more parameters [9].

2.1.1 Activation Potential

The Action Potential (AP) of a single cell is the changes that occur in a cell as the current waveform passes through. The exact chemical details that occur to support this change in the cell are outside the scope of this paper but more details can be found in Reference [3]. The AP changes depending on the type of cell that is being looked at and where during the wave's propagation the cell is located. However, there are three distinct phases of an AP signal caused by the waveform passing through that are most important: depolarization (or upstroke), plateau, and repolarization [3, 9]. An example of an AP for a cell is shown in Figure 2.1. For the AP shown, the transmembrane voltage (V) is shown along the vertical axis while time (t) is along the horizontal axis.

The depolarization phase occurs in the beginning of the waveform when the potential across the membrane becomes more positive than negative and opens the "gates" allowing a flow of charge to pass along the cell. This causes an increase in the cell's potential as seen during the rising region. The upstroke action normally occurs fairly quickly. Once the cell has become depolarized, it starts the process of repolarization. This early phase of repolarization is called the plateau phase because of the relatively slow changes that take place during it. During the plateau phase, the cell remains relatively positive. Finally, the plateau phase is terminated by the quick action of the repolarization phase during which the voltage difference drops quickly and the cell returns to the resting potential [3].

It is expected that these three phases may not show up exactly as they are in the sample AP. They instead vary depending on the properties of the cell and many other details which are discussed in Reference [3]. However, the sample AP does clearly demonstrate what these three phases could look like.

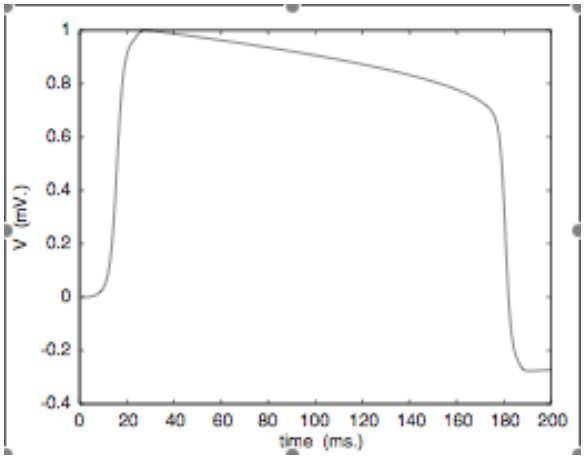


Figure 2.1: An example activation potential (AP) that clearly demonstrates the three phases of the waveform propagation

2.1.2 Ionic Current Models

There have been a variety of different models developed over the years that are intended to describe the ionic current of the cells. Some are simplified and meant for ease of use and others are meant to be very detailed and exact [4, 7, 9]. As cited in [8] and references therein, DiFrancesco and Nobel proposed a very complex model with over 50 parameters that results in the final system having about a dozen nonlinear equations. A more simplified model that only relies on V is purposed by Walton and Fozzard. Walton and Fozzard's model for the ionic current component is expressed as

$$I_{ion} = AV(1 - \frac{V}{V_t})(1 - \frac{V}{V_p}) \, ,$$

where A is a scaling factor, and V_t and V_p are the potential threshold and plateau value, respectively [8]. This model for the ionic current is able to produce a very basic example of the required AP as discussed in Section 2.1.1.

The Fitzhugh-Nagumo (FHN) model is more realistic for the ionic current and has been used and is described by many authors working on the Bidomain problem including [3, 4, 9]. The FHN model uses the transmembrane potential as well as introduces the recovery variable: w. The FHN model is commonly used today and has a couple variations [3, 4, 9]. The FHN model variable that is being used for this Bidomain implementation can be expressed as

$$\frac{\partial V}{\partial t} = \nabla \cdot (D \nabla V) + cV(V-1)(\alpha - V) - w \, ,$$

$$\frac{\partial w}{\partial t} = \epsilon(V - \gamma w) \, ,$$

where ϵ, γ, α, and c are all constants [9]. The FHN model, while greatly simplified from the original Hodgkin-Huxley formulation, it is still able to produce the three required phases of the AP [9].

2.1.3 Boundary Conditions

Many of the implementations today such as those of [4, 9], make the assumptions that there is not going to be any conductance from the tissue sample of interest to the surrounding area. Having boundaries that are closed and sealed is a reasonable assumption because the potential, V, is a constant value before and after the waveform has passed and the cell has recovered. These boundary conditions can be expressed as $\partial V/\partial n = 0$ and $\partial U/\partial n = 0$ [9].

2.1.4 Initial Conditions

It is assumed that the entire tissue sample is at rest at the start of the simulation. For the Bidomain model, using the Fitzhugh-Nagumo ionic current model, the intracellular, extracellular, and recovery potential at each discretized point starts at zero. These starting conditions are summarized mathematically in Equation 2.1. The method for exciting the tissue sample initially happens by applying a current to a portion of the domain.

$$\text{(Initial Conditions)} \qquad \begin{cases} V(x,y,0) = 0 \\ U(x,y,0) = 0 \\ w(x,y,0) = 0 \end{cases} \tag{2.1}$$

2.2 Program Performance Analysis

There any many techniques for judging the performance of a program in both serial and parallel cases. These techniques all differ depending on the application and type of statistics that are most relevant for their designed purposes. Two of those tools have been used here as a way of evaluating the performance of the linear system solver provided by PETSc. The tools and techniques employed for this analysis are completely independent of the specific problem and can be used to analyze many other program's. The goal of these tools, though, is to provide insight into a programs ability to use additional computational resources, (such as more memory or processors), efficiently.

2.2.1 Scalability

The first technique used when measuring the scalability of a program, is to look at the duration that is required to solve a given problem using a single processor and then again using many processors. When using multiple processors working together to solve for the solution to a single problem, the complexity depends greatly on the problem. Some problems may be distributed easily among groups of processors because each can do its own chunk of the work and then collaborate at the end, if any communication is needed at all. Other problems may need to pass data and intermediate results among all the processors involved. For these problems that require communication between all the processors, some overhead is required to manage those communications. Often times, if processors need to communicate, this must be done before it can continue performing work. Some processors may have to wait idle before getting back to work. All the extra time and resources needed to manage this interprocessor communication gets grouped together into the overhead of the parallel algorithm [5, 10, 12]. When there is a single processor working on a job, there is no need to track all

this extra information and therefore it is able to work purely on the job.

Speedup is a common technique for measuring the scalability of a program. Speedup is commonly defined as the time required for a given problem to be solved using a single processor over the time required to solve the same problem with multiple processors. For this definition, initially it is assumed that the all the input parameters, such as size, remain constant between the two trials [5, 10, 12]. Speedup is meant to measure the performance of an algorithm in serial versus parallel when trying to solve a problem using the respective durations. In general, parallel algorithms are attempting to stay close to the ideal speedup curve which represents the result where every processor added to the collective pool is being fully utilized with negligible overhead [5, 10, 13]. However, for many algorithms, this is a very hard result to achieve especially for fixed problems sizes. As the processor count grows, the amount of computation performed on each processor is decreasing while the communication may be constant or even growing. The speedup curve helps to show how well the work is being distributed among the various processors and how well as a whole they are able to effectively reduce the total duration required to find a solution to the problem.

In general, the speedup of a program is pretty loosely defined and is altered to reflect what test cases are feasible. For example, the problem size may be too large to be run on a single processor and therefore, the base or "serial" time needs to be the smallest processor count that the problem can be run on. An extension to the most basic definition of speedup is to use the same ratio but while changing the input parameters. A very common example of such an extension is the variability of the problem size. Using the original definition, this setup is considered to be strong scaling: the ability of the program to use more resources for a fixed set of input parameters and size. However, a second looser definition of speedup is classified as weak scaling where the problem size and other parameters are varied while keeping the processor count somewhat constant [13]. Many additional details about speedup analysis and program scalability are discussed in References [5, 10, 12, 13].

2.2.2 Communication and Computation Time

Part of understanding the performance of a parallel algorithm is defined by the portion of time spent performing computation for the problem and the overhead required to communicate and pass data. The communication and computation times are reflected in the speedup analysis, however, when analyzing the timing results, it can prove helpful to look at these two durations directly.

Part of analyzing the performance of a parallel algorithm looks more closely at the time required to perform any communication operations and the time spent performing the computation. In general, these timing classifications are going to be dependent on the problem. The problem that is being solved will control what operations need to be performed and thereby what algorithms can be used to accomplish that task. The ability of a given algorithm to efficiently use all computational resources will vary greatly depending again

on what operations are required. Once those elements are settled, varying the input parameters used when solving the problem will change the times as well. Using a single processor will eliminate the need for any communication and therefore will have a zero or negligible time compared to the computation. The problem size on the other hand will change how large of a portion of the problem a processor can be given. Typically a larger portion of the problem will require more time to perform the computational portion of the total time. Any and nearly all aspects of a given problem can influence the time required for these two categories.

Looking at the ratio of how much time is being spent performing the communication overhead associated with parallel algorithms versus serial algorithms can help to explain some of the speedup results. A reason for looking at this ratio is reflected in the fact that many times when communication tasks are taking place, the processor is forced to wait idle until the communication can complete. Commonly, the processor is able to execute instructions and perform operations much more quickly than they are able to pass data between each other. There are some techniques for reducing the time the processor is required to remain idle and waiting. These include overlapping the communication with the computation; performing both at nearly the exact same time. However, the success of these techniques will depend on the algorithms being used and the program as it is developed.

A desired result of the ratio of the communication over computation time would be as close to zero as possible. If this is the case, then it means that the time required to perform any communication operations is negligible compared to the time spent working on the problem. However, as the ratio gets close to 1, then there is an equal quantity of time being spent on both the communication and computational components. As a last case, as the ratio grows above 1, then there is more time being spent managing the communication operations than there is being spent performing the computations.

Chapter 3

Implementation Details

To assist with studying the scalability of the Scalable Linear System Solver (KSP) provided by the Portable, Extensible Toolkit for Scientific Computation (PETSc), two different problems are used. These two problems are the Poisson equation and the Bidomain model. Both problems use the linear system solver to find a solution. However, they each use a different algorithm that is better suited for the problem. Each problem is described in more detail in the following sections.

3.1 Poisson Equation

The Poisson equation has served as a good starting point for analyzing the performance of the linear system solver. The Poisson equation makes use of a similar approximation technique as the Bidomain model. Because the Poisson equation is linear, the system of equations needed to approximate the solution can be created and initialized manually. The system of equations established by the Poisson equation is also simplified from what the Bidomain model needs. The two dimensional discretization of the Poisson equation shown in Equation 3.1 uses the same spacial discretization as the Bidomain model.

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2_{i,j} + u_{i,j+1}}{h^2} = \alpha_{i,j} \tag{3.1}$$

For the discretization shown, h is the space between two mesh points and the indices i and j represent the x and y directions, respectively. However, because the Poisson equation consists only of a single PDE, it is easier to work with and to setup.

The source code used for this analysis of the Poisson equation has been provided as one of KSP examples. The code starts off by allocating and initializing the matrix and vectors needed to solve the approximation. For the Poisson problem in two dimensional space, the matrix system is very nicely structured and easy to create. Each row of the matrix is going to have at most five nonzero elements. Of those five nonzero elements, their values are one of the set $\{0, 1, -4\}$. Therefore, the matrix is easy to populate with the correct values. The matrix for a 2 x 2 mesh is shown in Equation 3.2 which estimates the solution at a total of four points. Once

the matrix has been created, the example program allows for the use of a random vector as the solution or it just defaults to a vector of ones. Once the actual solution has been established, the program calculates the right-hand side vector b from the linear system $Ax = b$. A third vector is then used to store the solution to the linear system as generated by the solver. After the system has been solved, the experimental solution is compared to the exact solution and checked for its accuracy.

$$\begin{pmatrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{pmatrix} \tag{3.2}$$

The Poisson example is using the Conjugate Gradient (CG) algorithm for solving the system. Because the matrix that is setup by the two dimensional finite difference approximation is symmetric, it is a perfect candidate to use the CG method [11]. For more details and a thorough introduction to the CG method, Reference [11] and references therein are a very useful starting point.

3.2 Bidomain Model

The original implementation of the code necessary to solve this Bidomain problem was done by Dr. Maria Murillo in 2002. Our work picked up from this point. At the time of the original work, the computational resources were much more limited than we have access to today. For the original work, Dr. Murillo only had access to a supercomputer with 128 processors each of which were running around 375 MHz. For the numerical computations required to solve the Bidomain problem, these limitations kept Dr. Murillo from using a tissue discretization finer than 512 x 512.

Today though, we have access to the IBM BlueGene/L supercomputer that is operated by the National Center for Atmospheric Research (NCAR), Frost which was funded by a National Science Foundation (NSF) grant. Using Frost, we now have access to a supercomputer with 1024 dual core nodes each operating at 700 MHz. Each processor has 512 MB of RAM that is shared between both cores.

The original code developed by Dr. Murillo and the code that is being used to perform our analysis are very similar but had to be modified to support the new goals. The details of those two code variations are described in the following sections.

3.2.1 Implementation History

The original code was developed by Dr. Murillo was part of her Ph.D research which was completed in 2002. There have been many changes made to the PETSc libraries between the start of our analysis and when

the original work was been performed. The code therefore needed to be updated to work with the most current version of PETSc. In the beginning, the process of updating the code was a bit slow due to not having access to the old documentation nor knowing exactly which version of PETSc was originally used.

After sending out a request for help to the PETSc-Users mailing list, we were pointed to where the full documentation sets from many previous PETSc releases could be found. With the old documentation, the process of updating the code started to pick up more. Soon a compiling and running version of the code had been achieved. During the process of updating the code, though, it was noticed that the equations as they were coded did not reflect the work described in the final results publication by Dr. Murillo. After a little bit of digging around through some old source files, a newer version of the original code was found where the equations were lining up much better with those described. Once again, this new source code was updated to reflect the interface changes of the most current version of PETSc (version 2.3.3 P5). There however were a couple small discrepancies between this version of the code and the work described in the final results: the code we had found was using the Walton and Fozzard ionic current model instead of the Fitzhugh-Nagumo model. Despite these small differences, we continued and got the code to a compilable and executable state.

Upon doing some initial analysis of the results produced, they were not lining up with what was described in the original publication. We spent some time looking through the code and the paper searching for a hint as to why the results we were getting were not matching. After correcting a few small details, we decided to move forward with implementing the Fitzhugh-Nagumo ionic current model. Once these new changes had been implemented, some testing was done to see if the results were now matching those described. However, there was still a hidden bug that was causing some problems in the results we were getting: the transmembrane voltage would continue to increase instead of dispersing throughout the tissue membrane and then dropping. After spending some time looking through the code and trying to isolate exactly where the problem might have been, it was decided to change the focus of the analysis. Instead of studying the ability of the algorithm as a whole to scale to larger processor counts and mesh sizes, we highlighted the fact that the majority of the computational time is being spent in the linear system solver. From here, we decided that it would be best to not spend additional time fiddling with trying to make the code work and reproduce the exact results, but instead to study the scalability of the core computational component.

3.2.2 Bidomain Discretization

In order to try and find a solution to the Bidomain problem, the equations need to be approximated using both spatial and time dependent approximations. The tissue sample under analysis is defined over a 1 cm $\times$ 1 cm region. We then use a uniform mesh ($\Delta x = \Delta y$) to define a set of points over this region at which we approximate the final solution. For the two spatial derivatives, we use the standard 5-point centered difference

stencil to approximate the second derivative at each internal discretized point. For the time derivative, we are using the fully implicit backward Euler approximation of the first derivative. Using this scheme to approximate the function, a large, sparse, non-linear system of equations can be defined as the resulting approximating. The new system can be solved at each time step to approximate the final solution.

3.2.3 Unique Solution Requirements

There are a couple small caveats to this problem however. The 5-point stencil described in Section 3.2.2 works well and is applicable to all the internal points of the mesh. The boundary conditions which were discussed in Section 2.1.3 describe how the current behaves when the edge of the tissue is reached. There are a couple points of this discretization that are unaccounted for: the four corners. These four points are not used in any of the approximation calculation. Even for the boundary conditions, they are updated by the other corners only. Therefore, they can be set to zero and help to isolate a unique solution.

3.2.4 Final System and Constants

Applying all the details described above including the ionic current and various discretizations, the full system is shown in Equations 3.3, 3.4, and 3.5. For these equations, i and j indicate the x and y position of a mesh point in the discretization, and n is the current iteration or time step. The subscripts on the various σ values, l and t, represent the conductivities in the longitudinal (x) and transverse (y) directions [7]. The superscripts on the various σ values, i and t, represent the intra- and extracellular domains, respectively. Finally, V is the transmembrane potential, U is the extracellular potential, and w is the recovery potential. The rest are all constants or physical parameters of the Bidomain model and associated components. The physical parameter values for this model come from References [4] and [9]. These parameters are all summarized in Table 3.1.

$$\begin{aligned} & c_m \frac{V_{i,j}^n - V_{i,j}^{n-1}}{\Delta t} - \sigma_l^i \frac{V_{i-1,j}^n - 2V_{i,j}^n + V_{i+1,j}^n}{(\Delta x)^2} - \sigma_t^i \frac{V_{i,j-1}^n - 2V_{i,j}^n + V_{i,j+1}^n}{(\Delta y)^2} \\ & \quad + cV_{i,j}^n(V_{i,j}^n - 1)(\alpha - V_{i.j}^n) - w_{i,j}^n \\ & \quad = \sigma_l^i \frac{U_{i-1,j}^n - 2U_{i,j}^n + U_{i+1,j}^n}{(\Delta x)^2} + \sigma_t^i \frac{U_{i,j-1}^n - 2U_{i,j}^n + U_{i,j+1}^n}{(\Delta y)^2} \\ & \quad + I_{app_{i.j}}^n \end{aligned} \tag{3.3}$$

$$\begin{aligned} & -(\sigma_l^i + \sigma_l^e)(U_{i-1,j}^n - 2U_{i,j}^n + U_{i+1,j}^n) - (\sigma_t^e + \sigma_t^e)(U_{i,j-1}^n - 2U_{i,j}^n + U_{i,j+1}^n) \\ & = \sigma_l^i(V_{i-1,j}^n - 2V_{i.j},^n + V_{i+1,j}^n) + \sigma_t^i(V_{i,j-1}^n - 2V_{i,j},^n + V_{i,j+1}^n) \end{aligned} \tag{3.4}$$

$$\frac{w_{i,j}^n - w_{i,j}^{n-1}}{\Delta t} = \epsilon(V_{i,j}^n - \gamma w_{i,j}^n) \tag{3.5}$$

Table 3.1: Physical Parameters and other required constants

$\chi = 2000/\text{cm}$	$c_m = 1.0\mu\text{F}/\text{cm}^2$
$\sigma_l^e = 4.0 \times 10^{-3}/\Omega\text{cm}^2$	$\sigma_t^e = 4.0 \times 10^{-3}/\Omega\text{cm}^2$
$\sigma_l^i = 4.0 \times 10^{-3}/\Omega\text{cm}^2$	$\sigma_t^i = 1.0 \times 10^{-3}/\Omega\text{cm}^2$
$c = 2.0$	$\alpha = 0.1$
$\gamma = 0.5$	$\epsilon = 0.002$

In the very beginning, the two domains and the transmembrane are considered to be at rest and therefore are all zero (at $t = 0$). The time step used for this implementation is $\Delta t = 1\text{ms}$ [9]. Finally the boundary conditions for the discretized system are shown in Equation 3.6.

$$\begin{aligned}
&(\text{lower edge}, y = 0), && \left\{\begin{aligned} F(V) &= V_{i,0} - V_{i,1}, \\ F(U) &= U_{i,0} - U_{i,1}, \end{aligned}\right\} && 0 < i < m \\
&(\text{upper edge}, y = 1), && \left\{\begin{aligned} F(V) &= V_{i,m} - V_{i,m-1},, \\ F(U) &= U_{i,m} - U_{i,m-1}, \end{aligned}\right\} && \\
&(\text{left edge}, x = 0), && \left\{\begin{aligned} F(V) &= V_{0,j} - V_{1,j}, \\ F(U) &= U_{0,j} - U_{1,j}, \end{aligned}\right\} && 0 < j < m \\
&(\text{right edge}, x = 1), && \left\{\begin{aligned} F(V) &= V_{m,j} - V_{m-1,j},, \\ F(U) &= U_{m,j} - U_{m-1,j}, \end{aligned}\right\} &&
\end{aligned} \tag{3.6}$$

3.2.5 Initial Exciting

As mentioned earlier, the tissue is at rest when $t = 0$ and is excited by applying a current to some portion of the domain. For this implementation, a current of $40 Amp/cm^2$ is applied to the center portion of the domain [9]. Other implementations have used this same initial current for exciting the tissue sample but have applied it in different regions [4].

3.2.6 Algorithm Overview

To summarize part of Dr. Murillo's work in Reference [7], the program creates a two dimensional mesh to represent the area over which the cardiac simulation will be performed. The simulation will approximate the solution at each of those mesh points. At each mesh point, there are three degrees of freedom: one for each equation (in the case of using the Fitzhugh-Nagumo ionic current model). However, when using the Walton-Fozzard ionic current model, each mesh point will only need to have two degrees of freedom: one for

the transmembrane voltage and one for the extracellular voltage. Once the mesh has been created by PETSc, it is divided into a collection of subdomains with one subdomain being assigned to each processor available. Once each processor has been initialized with its portion of the domain, a variety of vectors are created to hold the values for the approximation. From there, SNES is initialized with all the necessary settings and function pointer. At that point, SNES is ready to be used.

In order to solve the Bidomain model, the problem has to be solved in steps over the simulated time duration. The system is initialized to be all zeros and the time starts at $t = 0$. For this time step, an initial guess for the solution is zero. Once the solution has been solved for the first time step, the solution from the previous step is used as the initial guess for the next. The nonlinear system needs to be solved at each time step. In order to do this, SNES sets up a linear system for the current time step and then performs the necessary calculations to approach a final solution. The system is solved using the Generalized Minimal Residual (GMRES) method with the restricted additive Schwarz preconditioner. As part of the Schwarz preconditioner, the Incomplete LU (ILU) factorization is used to solve the subdomain problem. More details can be found in Reference [7].

3.2.7 New Implementation

In order to best study the underlying linear system of the Bidomain problem, it was easiest to allow the nonlinear system solver to perform the setup and initialize the linear system. Once the nonlinear solver has setup the underlying linear system, the matrix operator and right-hand side vector can be pulled out and analyzed separately.

To accomplish all of this, the nonlinear and linear solvers are limited to one iteration. That way they will start and be initialized but there were not be a significant delay waiting for either solver to terminate before moving on to the desired analysis. PETSc provides access to the linear solver context used as part of the nonlinear solver and through this direct access it is possible to pull out the matrix and constant vector. Copies of this matrix and vector are made to remove any dependence from the original linear solver. A new KSP solver is created and the matrix is assigned as being the system to be solved. Finally, any necessary tolerances and command line parameters are set.

Chapter 4

Software Tools

For the Bidomain problem, a software toolkit called the Portable Extensible Toolkit for Scientific Computing (PETSc) is being used [2]. PETSc is a software package that is distributed by Argonne National Laboratory and is a collection of C and Fortran routines that use the BLAS/LAPACK libraries to provide the necessary tools for solving systems of linear or nonlinear equations.

PETSc provides all the necessary components for the matrix and vector storage with data structures designed to optimize memory usage for both sparse and dense matrices. PETSc also provides the functionality to parallelize any required operations for which a parallel solution has been implemented. PETSc automatically manages all the data distribution between processors as well as communication necessary to complete any operations.

PETSc greatly simplifies the original code by off loading a huge chunk of the functionality to an already proven and reliable software library. All that needs to be implemented are the problem-specific components required for the system solvers. From there, PETSc handles the remainder of the operations and only makes calls to the specified user routines either for a function evaluation or for convergence testing as a couple examples.

The version of PETSc that was used with the original code is not known. However, after going through the process of updating the code so that it would work with the newest version, it could be guessed that somewhere around version 2.0.0 was being used. This guess comes from looking at the change logs and seeing when certain interface routines were changed and in which versions. For the updated code though, the most recent version of PETSc is being used to take advantage of algorithm and efficiency advances. The latest version of PETSc available in a stable state is version 2.3.3 patch 5.

The goal of analyzing the scalability of a program requires that a set of performance data be recorded during the program's execution. This performance data needs to summarize the characteristics of the running program and highlight how long as well as how many calculations were needed to complete a desired operation. A key to this data though is to not cause considerable overhead in recording the events or else the program's runtime characteristics will be skewed. A balance must be achieved between how much data needs to be saved

and being sure to exclude as much of the extra time required by the profiling operations as possible. A couple of the tools used to obtains these performance characteristics are described in the following section.

4.1 PETSc Profiling

To obtain much of the performance data necessary to do the scalability analysis, the logging tools have already been implemented as part of PETSc. PETSc provides a variety of different logging capabilities and utilities. It can be configured during the initial build process to include all the program profiling options or they can be turned off and not built into the compiled libraries. Having this option is helpful to reduce any extra overhead of making calls to performance gathering routines when the debugging information is not needed, will be ignored, or when full scale runs are being executed.

First, by using the **-log_summary** command line flag, PETSc will record many of the details about what is happening during the execution of the program. PETSc records run times, memory management operations, floating point operations per second (FLOPS) approximations, MPI messages that are sent and received, among many other statistics. It is also possible through this profiling interface to divide the recording of the data into different sections and thereby get statistics about chunks of code instead of just the entire program as a whole. Also, this option shows all the command line flags that were used, the number of processes assigned, when the job was run, and some PETSc debugging information as well. An important feature that the profiling interface does provide is the ability to run a chunk of code in its entirety multiple times saving the operational statistics about the last repeat only. Using this feature, most of the memory and communication lines will be allocated or opened in the first execution and therefore the overhead will not be present in the later repetitions. The profiling subset of PETSc is capable of providing many very useful statistics about the program.

4.2 Multi-Processing Environment

Another tool that has been used for obtaining profiling data is the Multi-Processing Environment (MPE) [1]. MPE is provided as an extension of the Message Passing Interface (MPI), both of which are developed and distributed by Argonne National Laboratory. PETSc uses MPI to provide the interprocess communications when multiple processes have been assigned for assisting with the calculations. MPE provides an interface to stamp and record events as the occur during execution. The programmer manually stamps when a certain part of the program has been reached or at the start of a routine and then again when the chunk of code or a routine has ended. This time-stamped data from each process involved is then all mixed together to produce a single log file with all the sequenced events.

MPE logging of events by PETSc is enabled by first configuring and building the PETSC libraries with debugging enabled and the MPE build flags. Once this has been done, the logging of the MPE data during

a running program can be performed by simply supplying the **-log_mpe** command line flag. This flag will signal PETSc to include all the time-stamped events and to save them to a file. Once the MPE events file has been generated, a set of Java tools can assist with analyzing the file by displaying the sequence of events for each process and communication between them. The utility for this is JumpShot and similarly is provided by Argonne. JumpShot displays in a graphical fashion the nested occurrence of events and how long they last as well as some of the communication that happens between processes.

Not every single routine of the PETSc libraries logs an event; instead only the higher level operations are recorded. These include most of the vector, matrix, solver, and operational routines. However, many of the routines that manipulate a small segment of data, perform a single calculation, or insert or access an element in a vector or matrix (for a couple examples) will all be ignored. If they were all to be recorded, the log files could potentially grow to huge sizes with a lot of information that would not be helpful for debugging. Also, the overhead required to log these events, while fairly minimal in comparison to many of the computational aspects, logging too many events would start to cause the overhead to escalate and become a impacting time factor.

Chapter 5

Numerical Results

The numerical simulations performed here have been designed to support obtaining performance results necessary to evaluate how well KSP is able to scale with more resources. Between the two problems investigated, the only changes that are made to the parameter sets are the algorithms being used by the linear solver. For the Poisson example, the Conjugate Gradient (CG) method is used while the Generalized Minimal Residual (GMRES) algorithm is used for the Bidomain problem. For the problems, the LU decomposition is used by the preconditioner. LU operates slightly different for the serial versus parallel cases. Ini serial, a direct solve is performed and therefore only a single iteration is needed. However, in parallel, more information is needed from the other processors to complete the local decomposition. Also in the parallel cases, the number of levels of overlap that are shared between the processors can be changed. The levels of overlap controls the number of rows that need to be shared by a processor with its neighbors. More levels of overlap typically result in fewer iterations but more data to be shared.

The other parameters that are varied are the processor counts, mesh sizes, and levels of overlap as part of the preconditioner. These three input parameters are all varied over the same sets of values to assist with consistency between the two problems. There is one small caveat regarding the mesh sizes between the Poisson and Bidomain problems. For the implementation we are using of the Bidomain problem, each point has two degrees of freedom and therefore the final matrix system will have twice as many rows for a given mesh size as the Poisson example. Finally, the processor counts used have been selected for a very specific reason. As part of both of these problems, the domain mesh is going to be broken up over all the processors available to assist with the calculations. All the mesh sizes used are equal in both dimension and therefore to avoid any conflicts, the processors counts are kept so that an even grid can be used to divide the mesh. For example, the processor count of 4 will produce a uniform 2 x 2 grid. Similarly, 16 processors produces a 4 x 4 grid. The mesh sizes have all been chosen as well for their ability to evenly divide among the processors available. The mesh sizes are all powers of two and therefore are evenly divisible many times.

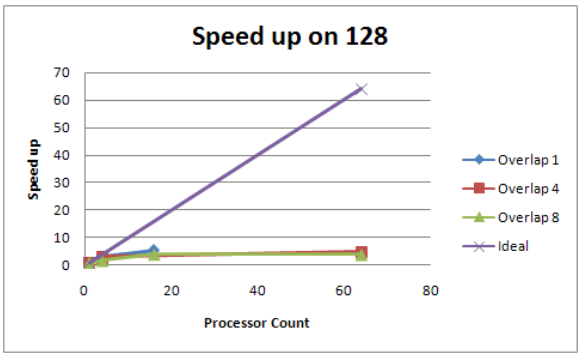
5.1 Poisson Results

For our analysis of the Poisson example, we tested it over a variety of different mesh sizes and processors counts. The Poisson example has been tested using three different mesh sizes: 128 x 128, 256 x 526, and 512 x 512. The domain for the Poisson example is defined over the region going from $(0, 1)X(0, 1)$. To solve the linear system, all of the Poisson test trials have been solved using the Conjugate Gradient (CG) method with the restricted additive Schwarz preconditioner using the LU factorization. The number of processors used is varied between 1, 4, 16, 64, and 256. However, for reasons that need to be examined more closely, certain test cases would not be solved before hitting the iteration limit of 10,000. While the cases that did not converge before this point were sporadic (occurring across all processor counts and different levels of overlap), the results for a given set of parameters was consistent across the three trials.

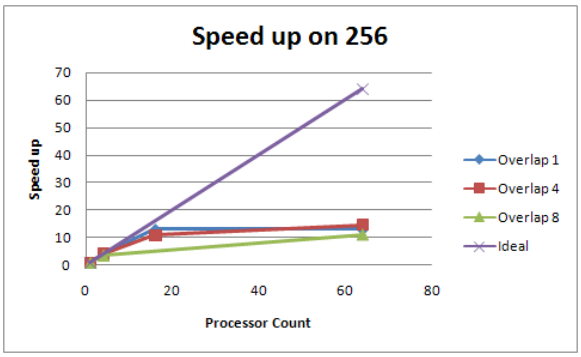
In general, a summary of the results of the Poisson example demonstrate that the KSP does struggle to maintain good speedup results and efficient usage of all processors when the number of processors starts to get big. However, this is dependent on the size of the mesh that is being used. In all the meshes that are being used for this analysis, each one started off by showing close to ideal speedups in the smaller processor counts and then deviated more and more as the processor count gets bigger. The important thing to note though is that the point at which this deviation starts to become significant varies and typically occurs with more processors for the larger mesh sizes.

To start, some of the first results that were looked at after compiling all the data from the PETSc profiling tools were the speedup curves. For the three different mesh sizes, the curves at first looked pretty bad once the ideal speedup was added. However, after chopping off the 256 process count test trials from the graphs and looking more closely at the trials between the counts or 1 and 64 processors, the speedups started to look a bit better. While the speedups do deviate from the ideal curve pretty quickly, there are still some test cases that do follow the ideal closely. These test cases that exhibit the near ideal speedups are all small processor cases. As the processor counts rise, the speedups drop off significantly. The speed up curves for the 128 x 128 and 256 x 256 mesh sizes can be seen in Figures 5.1(a) and 5.1(b). After looking over these figures, it becomes clear that the 256 x 256 mesh is performing closer to the ideal speedup longer. These results act as an initial assumption that the KSP using CG does not perform well for larger processor counts while using the smaller mesh sizes. Finally, the three levels overlap are all producing approximately similar speedup results.

As for the 512 x 512 speedups, initially the results look like they are performing worse as shown in Figure 5.2. A careful detail to notice though when comparing the 512 x 512 mesh with the other Poisson mesh sizes is that there is a missing point. The 512 x 512 mesh was too large to be run on a single processor and therefore the base speedup that is being used is the four processor test case. The four processor case requires less time



(a) 128 x 128 Mesh Speedup



(b) 256 x 256 Mesh Speedup

Figure 5.1: Speedup curves from the Poisson example

than the serial case so the time drops for using more processors are not as significant for the 512 x 512 mesh as is the case for the 128 x 128 and 256 x 256. Part of the reason for the relatively worse looking performance when using the four processor case as the base can be a attributed to algorithm differences in the serial versus parallel cases. For example, when using the LU factorization as part of the restricted additive Schwarz preconditioner, if only using a single processor then the factorization performs a direct solve method (which only requires a single iteration). However, when performing the LU factorization in parallel, the algorithm is forced to change. Despite these algorithm changes, it is still possible to see that the points displayed follow a close to linear relationship but the final point is a bit lower than the if it was perfectly linear. This dip shows that the 512 x 512 mesh case is able to use the additional processors more efficiently longer than either of the other meshes. It is important not to confuse the near linear relationship of the test cases of the 512 x 512 mesh with that of the ideal speedup. The 512 x 512 mesh speeds are clear lower than the ideal curve.

As we started to look more closely at the ratio of the durations spent performing communication and computation, it was helpful to look also at how many rows of the matrix are getting distributed onto each processor. The mesh is distributed in the most efficient manner across all processors available. As mentioned earlier, because the processor counts are all being kept as integer increment grid sizes, the mesh will be spread evenly across all processors. The number of rows of the matrix that are being assigned to each processor are shown in Table 5.1.

Table 5.1: Row division by processor and mesh size for the Poisson example

Processor Count	128 x 128	256 x 256	512 x 512
1	16384	65536	262144
4	4096	16384	65536
16	1024	4096	16384
64	256	1024	4096
256	64	256	1024

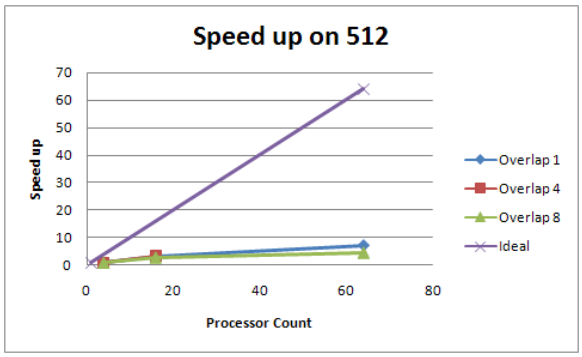


Figure 5.2: Speedup curve of the Poisson example using a 512 x 512 mesh and a 4 processor base case

Using the communication to computation ratios, it becomes easier to justify the reasons for why the speedups are dropping off quickly. However, using the ratio of the two components does not isolate exactly what part of the algorithm is the culprit in causing there to be as much time spent communicating. Instead, it just serves to re-emphasize the results of the poor speedup analysis. The ratios of these times are shown in Figure 5.3. Looking over this graph, it is clear that many of the trials are above a ratio of 1. For all of these trials, more time is being spent performing communication operations than is being spent performing the actual computation.

The graphs shown in Figures 5.4(a), 5.4(b), and 5.4(c), each show the percentage of time that is classified as communication or computation time during the KSP system solve operation. The graphs show the percentages for 1, 4, and 8 levels of overlap between the subdomains. As shown in Figure 5.4(a), the computation time starts off at 100% and negligible communication time for the one processor case. However, around 16 processors, they have already become equal: half of the total duration is spent communicating with the other processors and half spent performing calculations. Then for the next processor count, the two have completely crossed and more time is necessary for the communication than computation. When referenced against the row counts for each processor in the Table 5.1, when there are 64 processors, each one has very few rows to work with and many others to talk with.

Similar results are shown in Figures 5.4(b) and 5.4(c), for the 4 and 8 levels of overlap as part of the LU factorization, respectively. These two figures also show the same trends but they point at which the communication and computation times cross is getting moved towards higher processor counts. This is a sign that as more levels of overlap are specified, the processors do not need to communicate quite as much in order to complete their portions of the computation.

As a final note on the Poisson performance analysis, while the majority of the results provided have been for a 128 x 128 mesh, the results are very similar but less pronounced for the other two mesh sizes (256 x 256 and 512 x 512). Both of those mesh sizes display the same trend of requiring less communication time for

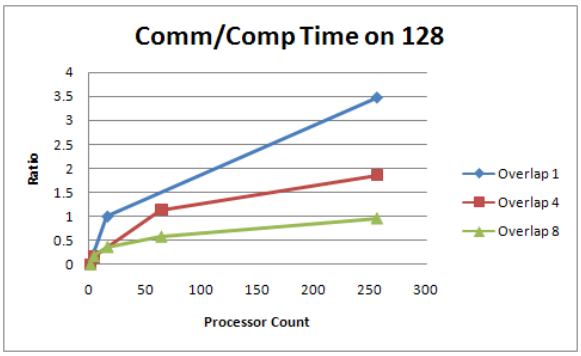
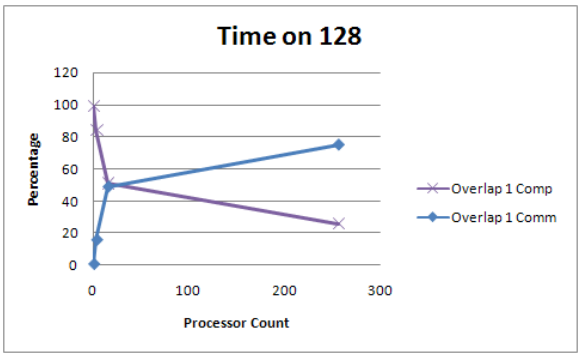
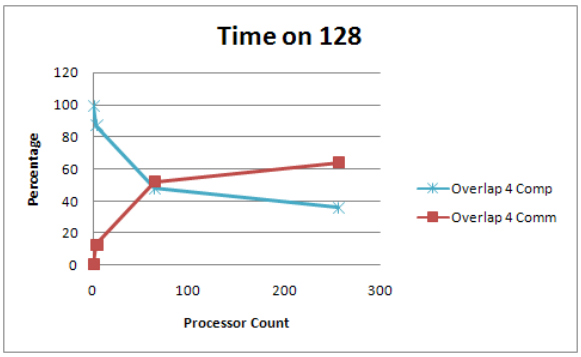


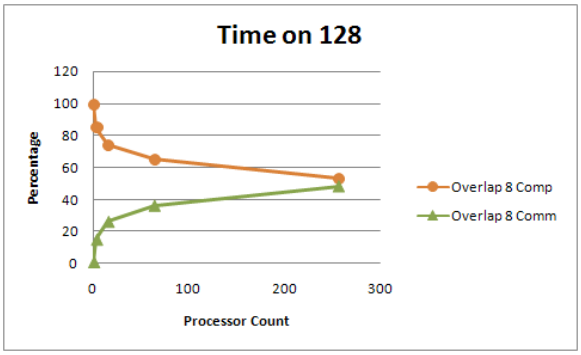
Figure 5.3: The ratio of communication time over the computation time for a 128 x 128 mesh using three different levels of overlap from the Poisson example.



(a) One Level of Overlap



(b) Four Levels of Overlap



(c) Eight Levels of Overlap

Figure 5.4: Communication and computation time percentages for the 128 x 128 mesh Poisson example using for the three different levels of overlap

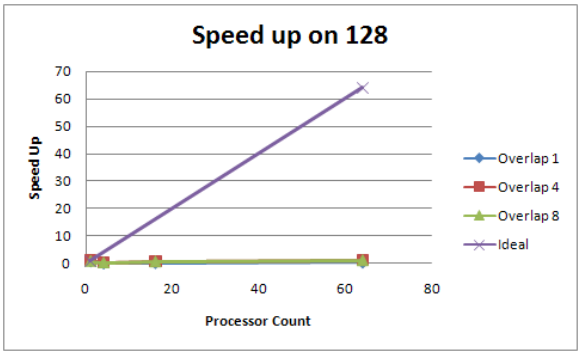
greater levels of overlap and tendencies towards crossing points. However, the results are much more clear for those that have been provided.

5.2 Bidomain Results

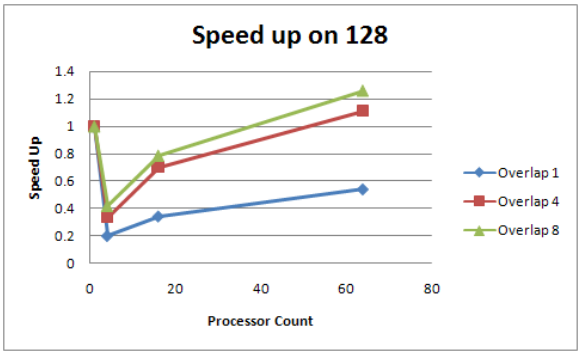
The results of the Bidomain problem look very similar to those shown and discussed for the Poisson when it comes to trends. However, the exact numbers and some other details set the two methods apart. Something to be aware of when analyzing the results of the Bidomain problem as compared to that of the Poisson: while the mesh sizes are the same, the generated linear systems do not have the same dimensions. The Bidomain problem has two data points at each mesh location that need to be approximated; one for each of the two equations. Having these two equations means that the matrix system that is generated as the approximation now has twice as many rows as the Poisson system did for the same mesh size.

To start off by looking at the speedup curves for the Bidomain problem, one of the first things that can be seen is that they are also very poor in comparison to the ideal speedup line as shown in Figure 5.5(a). However, these speed up curves are looking to be even a little bit closer to zero than was the case of the Poisson example. If the ideal curve is removed, a closer look at what is really happening for the speedup of the 128 x 128 mesh of the Bidomain problem can be seen as shown in Figure 5.5(b). The speedups dip at first and then slowly start to climb back up with only a couple of the test cases breaking above a speedup of 1. The dip in the speeds up reflects that the linear system solver is able to produce a solution faster time using a single processor than is possible using the parallel algorithms. When there is no need to communicate with the other processors because all the necessary data is stored on a single processor, the GMRES algorithm appears to be able to work more efficiently than 4 processors working together. However, this graph also shows that for the four and eight levels of overlap on the 128 x 128 mesh, once 64 processors are being used, the total time required to solve the solution is finally shorter than that of a single processor.

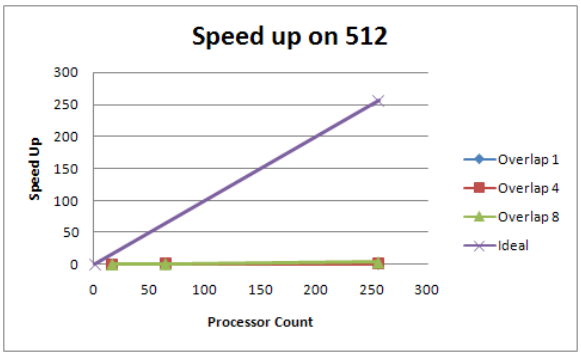
For the 512 x 512 mesh size, the speedup curve again looks like it is just a little bit above the axis along the bottom (Figure 5.5(c)). But when looked at again without the ideal curve, the tendencies of the lines can be better analyzed (Figure 5.5(d)). Similar to the Poisson, it was not possible to run the 512 x 512 mesh on a single processor and therefore, the shortest time of all trials using 4 processors becomes the baseline time for the speedup analysis. For eight levels of overlap, the speedup curve looks very linear but also this is not parallel to the ideal curve. While the speedup may follow a constant line, the speedup achieved is only 4 times greater when using 256 processors over only using 4 processors. As for the other two levels of overlap, their performance starts off looking good from the 4 to 16 processor cases. After that point though, both of these test cases run into a nearly zero speedup increase to the next increment of 256 processors. In general, the speedups are not maintaining anywhere near the ideal speedup for any of the mesh sizes that have been used.



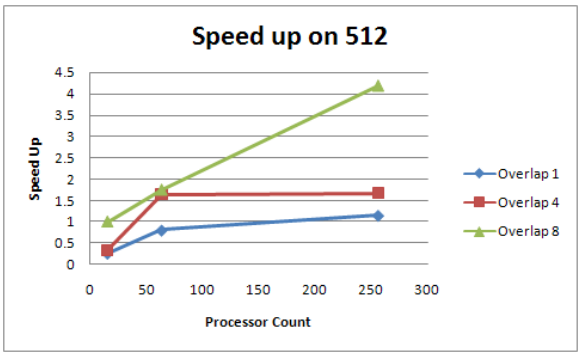
(a) 128 x 128 Mesh



(b) 128 x 128 Mesh without Ideal



(c) 512 x 512 Mesh



(d) 512 x 512 Mesh without Ideal

Figure 5.5: Speedup curves for the 128 x 128 and 512 x 512 mesh size test cases for the Bidomain problem

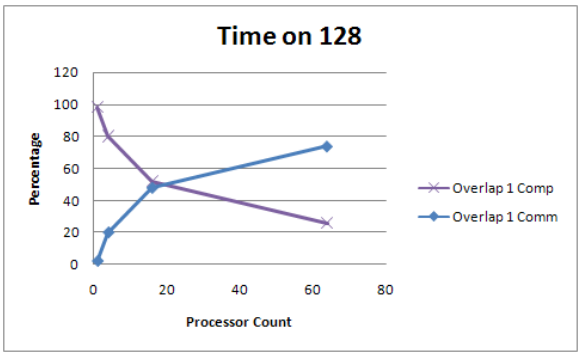
Table 5.2: Row division by processor and mesh size for the Bidomain problem

Processor Count	128 x 128	256 x 256	512 x 512
1	32768	131072	524288
4	8192	32768	131072
16	2048	8192	32768
64	512	2048	8192
256	128	512	2048

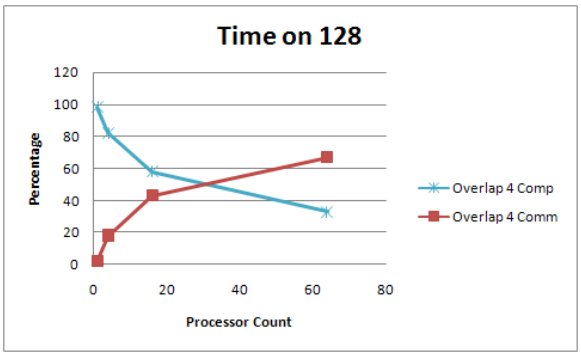
Turning to the more detailed analysis of the communication and computation time components, more details regarding why the performance results are coming out poorly can be seen. As with the speedups, the communication and computation times follow similar trends to those seen as part of the Poisson problem. However, a notable difference between the two algorithms that will be demonstrated, is that the GMRES algorithm appears to keep better performance in the smaller processor counts but is slower to overcome the communication time. To begin, Table 5.2 shows how many rows from the matrix are being assigned to each processor. As with the Poisson, all processor counts were kept so that the spread of rows would be evenly distributed.

For the 128 x 128 mesh, the percentages of time during the system solving classified as communication or computation time are shown in Figures 5.6(a), 5.6(b), and 5.6(c) for the three levels of overlap, 1, 4, and 8, respectively. Just as is the case of the Poisson example, as the number of processors increases, the durations of time spent in the two groupings become more equal and then eventually cross. However, the point at which they do cross is at a higher processor count for the 128 x 128 mesh with one level of overlap in the Bidomain problem than is the case in the Poisson example. Because these two durations require more processors to be used before the crossing occurs, the assumption can be drawn that the GMRES algorithm does not require as much communication as for CG. A second difference that can be seen between the two algorithms appears when looking across the time changes for the different levels of overlap. For the Bidomain problem, changing the levels of overlap used by the LU factorization does not have as significant of an impact on changing the times as it did for the Poisson example.

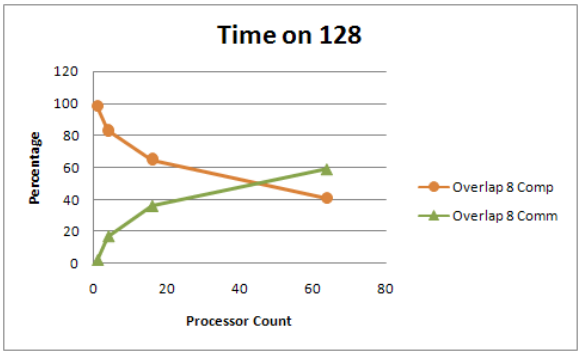
From the results, the Bidomain problem is not able to stop from having more communication time than computation time in the 256 processor trials until the mesh size reaches about 512 x 512 with 8 levels of overlap. This point of transition is contrasted with the Poisson where the communication time was not longer than the computation time for a 128 x 128 with 8 levels of overlap in the 256 processor test cases. Therefore, GMRES is much less responsive to changes in the mesh and overlap parameters. Much larger mesh sizes would be necessary to maintain a greater duration of computation time for the GMRES than is the case of the CG algorithm.



(a) One level of overlap



(b) Four levels of overlap



(c) Eight levels of overlap

Figure 5.6: Communication and computation time percentages for the 128 x 128 mesh size test case of the Bidomain problem

Chapter 6

Conclusion

In summary of the results, the KSP context provided as part of the PETSc library appears to struggle to maintain good performance as the processor count increases. The trend of poor performance is demonstrated in the results of both the Bidomain and Poisson problems. The appearance of poor performance is able to be masked through the use of a more finely defined discretization. Increasing the density of the mesh masks the communication time more than the smaller meshes by requiring more time be spent performing the computational operations. The communication time is not being decreased for the larger meshes but instead is growing slower than the computation time as the mesh increases. As the mesh size increases though, similar problems are experienced as with trying to run the 512 x 512 mesh on a single processor; the memory is not large enough. For example, using a 1024 x 1024 mesh for the Bidomain problem, a minimum of 16 processors can be used to store the entire matrix while keeping a square processor grid. On the Frost system, the memory size is becoming a factor impacting the scalability by setting an upper limit on the size of the mesh that can be used.

Looking more at the results of the Poisson and Bidomain problems, the speedups show that the Poisson problem can maintain better scalability than the Bidomain problem. By comparing the speedup curves for the 512 x 512 mesh size for the two problems, the results from the Poisson problem are slightly closer to the ideal speedup curve than for the Bidomain problem. The speedup curve for the 512 x 512 does not, however, show any results close to the ideal curve, even though the Poisson problem does perform relatively better than the Bidomain problem. Overall, the results point to signs of poor scalability for KSP under both problems. The performance of KSP can be dependent on the problem being solved. The Bidomain and Poisson problems both are similarly structured where both generate a sparse and banded matrix. However, the performance of KSP on other problems could vary depending on the problem and the specific matrix that is required. For these two problems, KSP does not maintain efficient usage of all processors for the larger mesh sizes. For the smaller processor counts through, each processor will maintain better efficiency despite long durations to solve the problem.

Bibliography

[1] MPE Web page, Mar 2008. http://www-unix.mcs.anl.gov/perfvis/.

[2] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, Mar 2008. http://www.mcs.anl.gov/petsc.

[3] Dekanski J Holden A Boyett M, Clough A. Computational Biology of the Heart, chapter 1. Modelling Cardiac Excitation and Excitability, pages 1–47. John Wiley & Sons, Ltd, 1997.

[4] J. P. Keener and K. Bogar. A numerical method for the solution of the bidomain equations in cardiac tissue. Chaos: An Interdisciplinary Journal of Nonlinear Science, 8(1):234–241, 1998.

[5] Vipin Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. Journal Parallel and Distributed Computing, 22(3):379–391, 1994.

[6] Valeria Simoncini Micol Pennacchio. Efficient algebraic solution of reaction-diffusion systems for the cardiac excitation process. Journal of Computational and Applied Mathematics, 145(1):49–70, Aug. 2002.

[7] Maria Murillo. Parallel Algorithms and Software for Time-Dependent Systems of Nonlinear Partial Differential Equations with an Application in Computational Biology. PhD thesis, University of Colorado - Boulder, 2002.

[8] Maria Murillo and Xiao-Chuan Cai. Computational Science - ICCS 2002, chapter Parallel Newton-Krylov-Schwarz Method for Solving the Anisotropic Bidomain Equations from the Excitation of the Heart Model, pages 533–542. Springer Berlin / Heidelberg, 2002.

[9] Maria Murillo and Xiao-Chuan Cai. A fully implicit parallel algorithm for simulating the non-linear electrical activity of the heart. Numerical Linear Algebra and Applications, 11(2-3):261–277, 2004.

[10] Peter S. Pacheco. Parallel Programming with MPI. Morgan Kaufmann Publishers, Inc., 1997.

[11] Jonathan Richard Shewchuk. An introduction to the Conjugate Gradient Method Without the Agonizing Pain. 1994. http://www.eas.asu.edu/ vasilesk/EEE598/painless-conjugate-gradient.pdf.

[12] J.P. Singh, J.L. Hennessy, and A. Gupta. Scaling parallel programs for multiprocessors: methodology and examples. Computer, 26(7):42–50, Jul 1993.

[13] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A framework for performance modeling and prediction. Supercomputing, ACM/IEEE 2002 Conference, pages 21–21, 16-22 Nov. 2002.

[14] J.P. Whiteley. An Efficient Numerical Technique for the Solution of the Monodomain and Bidomain Equations. Biomedical Engineering, IEEE Transactions on, 53(11):2139–2147, Nov. 2006.